Corndel Digital.

in association with

Softwire

The Corndel DevOps Engineering Programme

in association with Softwire

Module 1 – General Purpose Programming Workshop

Agenda

1000	Welcome and introductions
	Quick reminder of how workshops will run
	Part 1: Understanding basic Python • Recap + New Stuff (15 min) • Exercise (1 hour 30 min)
1200	**Lunch Break (1 hour)**
1300	**Part 2: Understanding version control with Git** • Recap + New Stuff (15 min) • Exercise (50 min)
1410	**Part 3: Understanding of HTTP and web apps** • Recap + New Stuff (15 min) • Exercise (1 hour 20 min)

How the Workshops Will Run

- Will typically involve a recap of core reading material
- Some new concepts required for the exercises will be introduced
- Exercises will typically be carried out in pairs/groups
- Workshop format may change slightly over time and depending on the module

Objectives of this Workshop

- Reinforce understanding of coding fundamentals in Python
  - Familiarity with a programming language will be necessary for writing API integrations & automated tests (covered in Modules 2 & 3 respectively)
- Build familiarity with common git commands, branching & raising Pull Requests
  - Source control underpins future workshop exercises and allows tutors to easily review project exercise work (through Pull Requests)
- Obtain a grasp of basic web app concepts (using Flask)
  - Setting up a web server will give us early experience with the HTTP protocol (which we'll cover in detail in Module 2)

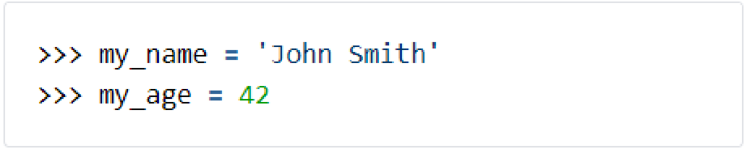
Basic Python - Mentimeter

Part 1: Understanding Basic Python

Recap

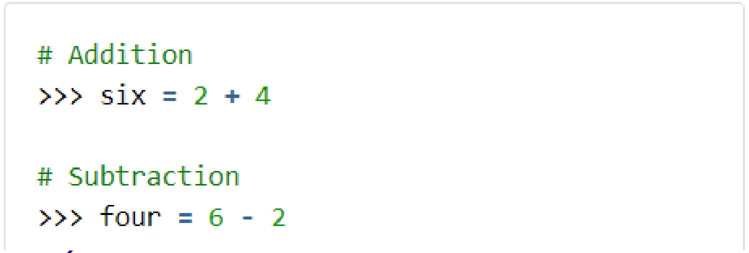
You should already know this, but as a reminder:

How you assign variables



```
>>> my_name = 'John Smith'
>>> my_age = 42
```

How you perform operations (such as arithmetic)



```
# Addition
>>> six = 2 + 4

# Subtraction
>>> four = 6 - 2
```

- Basic Data Types, with operations allowed on each:

```
>>> first_string = 'Hello'
>>> second_string = 'World'
>>> first_string + second_string # 'HelloWorld'

>>> na = 'Na'
>>> batman = 'Batman'
>>> 8 * na # 'NaNaNaNaNaNaNaNa'
>>> 8 * na + batman # 'NaNaNaNaNaNaNaNaBatman'
```

```
>>> a_string = 'Example'

>>> a_string[0] # E
>>> a_string[1] # x
>>> a_string[6] # e

>>> a_string[-1] # e
>>> a_string[-3] # p
```

- Different types of flow in Python:

```
if username == 'user':
    print('Hello, User')
    if password == 'squirmbag':
        print('Access granted')
    else:
        print('Access denied')
```

```
times_run = 0
while times_run < 10:
    print('Hello!')
    times_run = times_run + 1
```

- Functions Definitions & Usages:

```python
def print_item(name, price_in_pennies):
    formatted_price = '£{:.2f}'.format(price_in_pennies / 100.0)
    print('Item: ' + name)
    print('Price: ' + formatted_price)

print_item('Milk', 85)
print_item('Coffee', 249)
print_item('Orange Juice', 110)
```

- Complex Data Types (Such as Lists & Dictionaries):

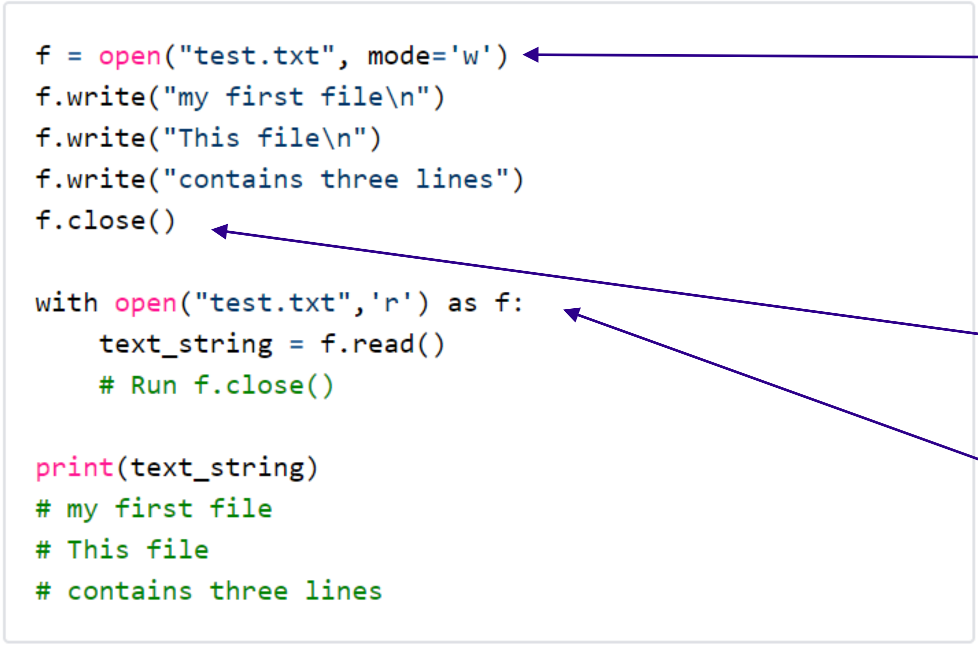
```python
>>> list_of_numbers = [3, 1, 4, 5, 7, 2, 6]
>>> list_of_numbers.remove(4)
>>> list_of_numbers # [3, 1, 5, 7, 2, 6]

>>> list_with_repeats = [1, 2, 1, 3, 2]
>>> list_with_repeats.remove(2)
>>> list_with_repeats # [1, 1, 3, 2]
>>> list_with_repeats.remove(1)
>>> list_with_repeats # [1, 3, 2]
```

```python
>>> favourite_colours = {}
>>> favourite_colours['Alice'] = 'Purple'
>>> favourite_colours['Bob'] = 'Green'
>>> favourite_colours # {'Alice': 'Purple', 'Bob': 'Green'}
```

Part 1: New Concepts

Basic File Operations

Python provides various built-in functions for handling reading/writing of files:



```
f = open("test.txt", mode='w')
f.write("my first file\n")
f.write("This file\n")
f.write("contains three lines")
f.close()

with open("test.txt",'r') as f:
    text_string = f.read()
    # Run f.close()

print(text_string)
# my first file
# This file
# contains three lines
```

The 'mode' keyword argument let's us specify whether we're expecting to write ('w'), read ('r') or otherwise. [See the docs for more options]

It's good practice to explicitly close file handle after you're done with it – but it's easy to forget!

To simplify that, the 'with' syntax here will handle that for us; at the end of the block it will automatically close the file.

Tuples

Creation:

```
my_tuple = 1, 2, 3
my_tuple = (1, "Hello", 3.4)
my_tuple = ("mouse", [8, 4, 6], (1, 2, 3))
print(my_tuple[0])
# "mouse"
```

Can be used to return multiple values from functions:

```
def divide_with_remainder(x, y):
    return x // y, x % y

output = divide_with_remainder(13, 5)
print(output)
# (2, 3)
value, remainder = divide_with_remainder(18, 7)
print(f"{value}, {remainder}")
# 2, 4
```

List Comprehensions In Python

Format:

```
[expression for item in list]
```

Simple Example:

```
h_letters = [ letter for letter in 'human' ]
print(h_letters)
# ['h', 'u', 'm', 'a', 'n']
```

More Complex Examples:

```
squared_odd_numbers = [ x*x for x in range(20) if x % 2 != 0 ]
print(squared_odd_numbers )
# [1, 9, 25, 49, 81, 121, 169, 225, 289, 361]
```

```
pythagorean_triples = [
    (x, y) for x in range(1, 10)
    for y in range(1, 10)
    if math.sqrt(x*x + y*y).is_integer()
]
print(pythagorean_triples)
# [(3, 4), (4, 3), (6, 8), (8, 6)]
```

Part 1: Exercise

The Ultimate Goal

The steps on the next slides will lead to a Python script that can process a file with a list of instructions:

```
goto 4

replace 1 2

remove 3

goto 2

goto calc x 3 5

replace 6 10
```

Step 1:

Write a basic Python "calculator".

It should accept 3 pieces of input from the user: a string that's one of "x", "+", "-", or "/" (an operation), an integer (parameter A), and another integer (parameter B).

It should then emit the result of performing the operation on A and B.

For example, if your application asks the user for an operation and 2 numbers, and the user enters "+", "1", "2", then the application should output "3".

If the user supplied "/", "5", "2", the application should output "2.5".

If the user supplied "x", "5", "0", the application should output 0.

Step 2:

Next process the following file: Link
Each line contains a calculation statements prefixed by "calc":

```
calc x 2 5
calc / 10 5
```

Create a new Python script, and within it:

- Read in the new file (see the earlier slide!)
- Compute the value of each line using the approach from step 1
- Add up the results from all the lines and send the results to the trainer

Hint 1: For reading the lines from the file you may want to use file.read().splitlines() *to build a list of lines.*
Hint 2: you may want to use string.split() to break up the parts of each calc line.

Step 3 (Page 1 of 2):

Next navigate the following file: Link

This has goto statements like the following

```
goto 27
```

This means go to line 27 in the file and read the statement there. Please note that calc and goto statements can be combined like so:

```
goto calc / 27 9
```

This is equivalent to `goto 3.`

For simplicity assume that we cannot nest calc statements, decimals are rounded down and out of bounds gotos (i.e. invalid line numbers) do not occur.

Step 3 (Page 2 of 2):

Starting from line 1, use the rules above to navigate the document, stopping when you've **hit a statement you’ve seen once before (they are allowed to be from different lines!)**.

When finished please send the statement and line number the code has stopped on to a tutor – but feel free to carry on and start the next stage while you wait for confirmation.

Step 4 (Page 1 of 2):

Finally navigate the following file: Link

This has some additional statements.

The goal is to process the file, starting from line 1, stopping when you’ve **hit a statement you’ve seen before or manage to jump outside the file by a goto**.

When finished, please send the line number & statement to the trainer to confirm.

Step 4 (Page 2 of 2):

Additional Statements:

```
remove {line_number}
```

- Remove line {line_number} from the file (if the line number does not exist do nothing) and then
- Read the next instruction after this remove statement

```
replace {line_number_1} {line_number_2}
```

- Replace line {line_number_1} with line {line_number_2} (if either line number does not exist do nothing) and then
- Read the next instruction after this statement

Lunch

Back at 13:05